JavaScript API Deprecation in the Wild: A First Assessment

Romulo Nascimento, Aline Brito, Andre Hora, Eduardo Figueiredo
Department of Computer Science
Federal University of Minas Gerais, Brazil
{romulonascimento, alinebrito, andrehora, figueiredo}@dcc.ufmg.br

***Abstract*—Building an application using third-party libraries is a common practice in software development. As any other software system, code libraries and their APIs evolve over time. In order to help version migration and ensure backward compatibility, a recommended practice during development is to deprecate API. Although studies have been conducted to investigate deprecation in some programming languages, such as Java and C#, there are no detailed studies on API deprecation in the JavaScript ecosystem. This paper provides an initial assessment of API deprecation in JavaScript by analyzing 50 popular software projects. Initial results suggest that the use of deprecation mechanisms in JavaScript packages is low. However, we find five different ways that developers use to deprecate API in the studied projects. Among these solutions, deprecation utility (i.e., any sort of function specially written to aid deprecation) and code comments are the most common practices in JavaScript. Finally, we find that the rate of helpful message is high: 67% of the deprecations have replacement messages to support developers when migrating APIs.**

***Index Terms*—API deprecation, JavaScript, Software Library**

I. Introduction

Building an application using third-party libraries is a common practice in software development. Libraries provide reusable functionality to client applications through their Application Programming Interfaces (APIs). API usage brings several advantages to a software development project [10], such as cost and resources usage reduction. As a result, developers can focus on business core requirements and software quality may increase by relying on libraries that have been widely adopted, tested and documented [6].

As any other software system, libraries and their APIs evolve over time [5]. Thus, functions and parameters might be renamed, updated, moved, or removed. Consequently, client applications need to migrate to the latest stable versions of their dependencies [1]. To help version migration and ensure backward compatibility, a recommended practice in software development is to deprecate the API. In other words, deprecation indicates that the use of a certain API should be avoided because it will be changed, removed or discontinued in a future version [7]. Some of the most popular programming languages, such as Java and C#, provide native support mechanisms and tools to help developers explicitly deprecate their APIs [9]. Indeed, recently, there have been many research on deprecation practices and mechanisms mostly on those languages [1], [4], [7], [9], [13]–[16]. However, to the best of our knowledge, there are no detailed studies regarding API deprecation in the JavaScript ecosystem.

JavaScript has become extremely popular over the last years. According to the Stack Overflow 2019 Developer Survey[1], JavaScript is the most popular programming language in this platform for the seventh consecutive year. GitHub also reports that JavaScript is the most popular language in terms of unique contributors to both public and private repositories[2]. The npm platform, the largest JavaScript package manager, states on their latest survey[3] that 99% of JavaScript developers rely on npm to ease the management of their project dependencies. This survey also points out the massive growth in npm usage that started about 5 years ago. Despite the growth on the usage of JavaScript external libraries and APIs, little is known about JavaScript API deprecation mechanisms and practices.

Therefore, this paper aims to investigate deprecation in JavaScript APIs. We analyze deprecation solutions in this language by inspecting the most popular projects hosted on npm. The following research questions are then proposed:

- **RQ1**. *How do developers deprecate JavaScript APIs?*
- **RQ2**. *Are JavaScript APIs deprecated with replacement messages?*

Replacement messages (RQ2) are important because they help developers to find alternative options for a deprecated function. Our initial results suggest that the use of deprecation mechanisms in JavaScript packages is low. However, we find five different ways that developers use to deprecate API in the studied projects. Among these solutions, deprecation utility (i.e., any sort of function specially written to aid deprecation) and code comments are the most common practices in JavaScript. Finally, we observed that the rate of helpful message is high: 67% of the deprecations have replacement messages to support developers when migrating APIs.

II. The JavaScript Ecosystem

JavaScript (or JS, for short) is a versatile programming language that conforms to the ECMAScript specification. It has been primarily designed and known as a language for rendering dynamic content on the client-side of Web applications. More recently, JavaScript has also been used as a server-side language through the use of the Node.js platform.

[1]https://insights.stackoverflow.com/survey/2019
[2]https://octoverse.github.com
[3]https://javascriptsurvey.com

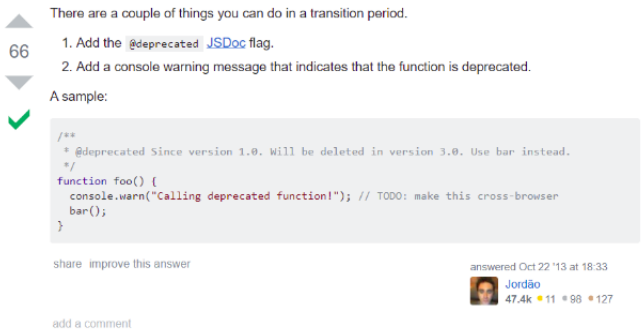

Fig. 1. Stack Overflow answer in which the author recommends using JSDoc annotation with a console warning message to indication deprecation.

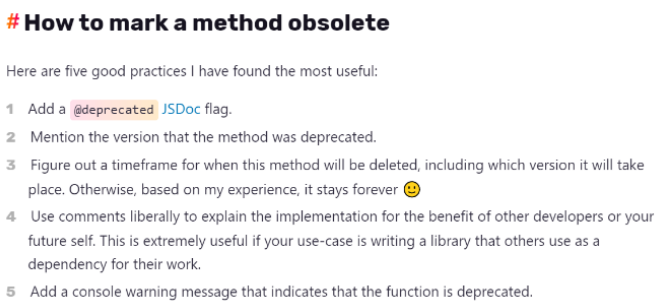

Fig. 2. Web development blog article suggesting the use of JSDoc, comments on the deprecation context and time frame, and console warnings.

Software reuse has become a key factor for a cost and time efficient software development project [11]. This scenario has led to the emergence of software repositories, such as npm, that provide a centralized and simplified management and distribution of software components. The npm registry serves as a base for JavaScript Web applications, frameworks and library ecosystems. On June 4th, 2019, npm reached one million hosted JavaScript packages, making it the largest software repository to date [12].

Unlike other popular programming languages, such as Java and C#, JavaScript provides neither native deprecation mechanisms, nor recommendations from ECMA International or TC39 on how to properly deprecate JavaScript written code. Google brings only a few relevant results when we search for "*how to deprecate JavaScript*". There is an popular answer to a Stack Overflow question, presented in Figure 1, in which the author recommends using JSDoc[4] deprecation annotation, possibly along with a console warning message indicating the deprecation[5]. In addition, Figure 2 shows a Web blog that also endorses the use of JSDoc – with comments on the deprecation context and time frame – and console warnings. The same blog also suggests that a deprecation utility, such as a helper function, might be suitable[6]. These recommendations are among the top results provided by a Google search. However, from our ad hoc searches, we observed that there is no standard way for API deprecation in JavaScript.

III. Study Design

This section describes the methodological steps we followed to answer the research questions presented in Section I. We first present the dataset of projects we used and the search strategies to find API deprecation in the target projects.

A. Selecting Projects

We selected the top-50 most depended upon projects according to the npm registry[7] to compose our dataset. npm

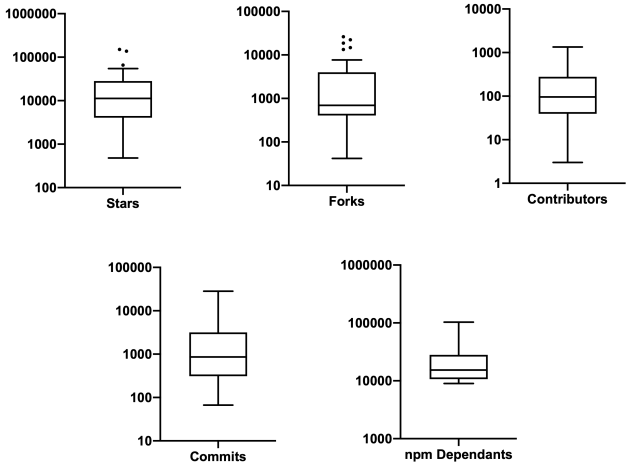

Fig. 3. GitHub projects statistics and npm dependents

is a well known package manager for JavaScript applications, which is a public collection of open-source JavaScript projects. Therefore, the npm registry website is an indicator of project popularity and their amount of client applications. To identify the characteristics of these JavaScript projects, we also collected metrics from their GitHub repositories. Table I presents all selected projects by showing their names and used versions. As shown in this table, our dataset includes some popular JavaScript projects, such as AngularJS, JQuery, and React.

Figure 3 shows some statistics about these projects based on GitHub and npm data retrieved in November, 2019. In particular, Figure 3 presents boxplots with the number of stars, forks, contributors, commits, and dependent clients. We collected the first four metrics from GitHub, while the last one was obtained from npn. As can be observed in this figure, the selected projects are not only highly popular (e.g., median of 10K stars), but also forked a lot. They are also active and with thousands of dependent clients. In fact, according to npm, all selected projects have more than 10K dependents.

B. Searching for Deprecation Occurrences

We downloaded the source code of the 50 selected projects, considering their latest stable version on November 20th, 2019. We then used a regular expression based search to find

[4]https://jsdoc.app/
[5]https://stackoverflow.com/questions/19412660/how-should-i-mark-a-method-as-obsolete-in-js
[6]https://css-tricks.com/approaches-to-deprecating-code-in-javascript/
[7]https://www.npmjs.com/browse/depended

TABLE I
50 MOST DEPENDED UPON PACKAGES ON NPM REGISTRY

	Project Name	Version		Project Name	Version
1	angular	8.2.10	26	moment	2.24.0
2	async	3.1.0	27	node-fetch	2.6.0
3	aws-sdk-js	2.550.0	28	node-fs-extra	2.1.2
4	axios	0.19.0	29	node-glob	7.1.4
5	babel	7.6.4	30	node-mkdirp	0.5.1
6	babel-loader	8.0.6	31	node-semver	6.3.0
7	bluebird	3.7.1	32	node-uuid	3.3.3
8	body-parser	1.19.0	33	prop-types	15.7.2
9	chalk	2.4.2	34	q	2.0.2
10	cheerio	0.21.0	35	react	16.10.2
11	classnames	2.2.6	36	react-redux	7.1.1
12	colors.js	1.4.0	37	redux	4.0.4
13	commander.js	4.0.0-1	38	request	2.88.1
14	core-js	3.3.2	39	rimraf	3.0.0
15	css-loader	3.2.0	40	rxjs	6.5.3
16	debug	4.1.1	41	shelljs	0.8.3
17	dotenv	8.1.0	42	style-loader	1.0.0
18	eslint	6.5.1	43	through2	3.0.1
19	express	4.17.1	44	tslib	1.6.0
20	generator	4.1.0	45	TypeScript	3.6.4
21	Inquirer.js	6.0.0	46	underscore	1.9.1
22	jquery	3.4.1	47	vue	2.6.10
23	js-yaml	3.13.1	48	webpack	4.41.2
24	lodash	4.17.15	49	winston	3.2.1
25	minimist	1.2.0	50	yargs	14.2.0

deprecation occurrence candidates. The regular expression we used on the search was '(@)deprecate(d)' since that covers any occurrences of 'deprecate', 'deprecated', and '@deprecated'. This last case is relative to the JSDoc annotation. We also tried other terms, such as 'obsolete' and 'replacement', but they returned very few relevat results. Next, we developed a tool to navigate through JavaScript files within all projects and find any occurrences of the deprecation regular expression on their source code. It is important to mention that we only consider main source code files, excluding test, minified, and non JS files (e.g., CSS and HTML). Every time one or more matches were found on a file, the file path and the code snippet were saved for further investigation.

C. Data Analysis

To support our analysis and the identification of API deprecation candidates in all 50 projects, we also used a JavaScript code parsing library, Flow[8], to find the context in which the API deprecation terms occur. We then exported the generated abstract syntax trees (ASTs) to manually analyze the deprecation occurrences. The abstract syntax trees previously obtained from the parser tool and the respective code snippets were used as input for a manual analysis. To find out how the deprecation terms were used, we sampled 20% of the deprecation occurrences. Finally, in the last step of our analysis, we categorize each API deprecation candidate.

Table II shows the five possible JavaScript deprecation cases we found in our analysis. We empirically derived this cases by manually and carefully analysing the samples of code snippets. If a certain occurrence does not fall in one of the proposed JavaScript deprecation mechanisms, we classify it the Others category. This analysis was performed by the first author of the paper and discussed with co-authors until consensus was achieved. We also paid special attention to verify whether the deprecation occurrences included replacement messages to help answering RQ2.

TABLE II
JAVASCRIPT DEPRECATION MECHANISMS.

JS Deprecation Mechanism	Description
JSDoc	Use of the @deprecation JSDoc annotation
Code comment	Use of code comments excluding occurrences of JSDoc
Deprecation utility	Any sort of code function specially written to aid code deprecation at any complexity level
console.*	Use of the JavaScript engine native console API
Deprecation lists	List of deprecated elements
Others	Other adopted solutions

IV. RESULTS

We found deprecation occurrences in 29 (58%) out of the 50 projects. At the file level, from 7,038 JavaScript parsed files, we detected deprecation occurrences in 214 (3%). The parsing tool extracted 1,279 deprecation contexts from the 214 files analyzed. From those, we selected a random sample of 268 cases (20%) for manual analysis.

We observed that the aws-sdk-js project alone represented about 25% of all found occurrences and that it could bias the results. To better understand the impact of this project on results, we compared results with and without the aws-sdk-js project. We concluded that the difference between the results considering the aws-sdk-js project and not considering it is not statistically significant.

As presented in Figure 4, the most frequent deprecation mechanism is *deprecation utility*. Deprecation utility is any sort of code function specially written to aid code deprecation. This case represented 88 (32.8%, ±5.6% for a 95% confidence level) out of 268. From those 88 occurrences, we detect that 75 contain replacement messages to support API migration. By analyzing the implementation of those deprecation utilities, we detect that 77 adopt local solutions to deprecate APIs, while 11 rely on third-party libraries. For example, a popular third-party often adopted to deprecate APIs in JavaScript is *depd*[9]. Interestingly, from the 77 local solutions, we observed that 64 throw warning messages to the console, 12 throw console errors, and 1 uses console traces to flag deprecation.

Deprecation indicated by *code comments* represent 27 (10.1%, ±3.6% for a 95% confidence level) of the cases. This represents the usage of code comments excluding occurrences of JSDoc. Only 4 of those code comments contain replacement messages. Additionally, 20 out of the 27 comments refer to the deprecation of API elements within the project, while 7 refer to the usage of deprecated external dependencies.

[8]https://flow.org/

[9]https://www.npmjs.com/package/depd

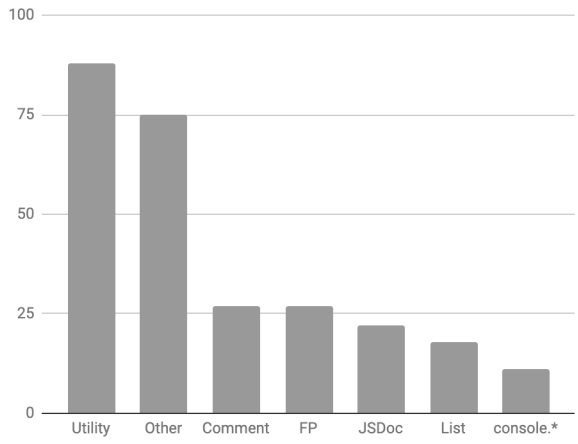

Fig. 4. Deprecation mechanism occurrences per category.

The adoption of the *@deprecated JSDoc* annotation was identified 22 times (8.2%, ±3.3% for a 95% confidence level). However, only 10 of those occurrences have replacement messages. Deprecation elements described trough *lists* represent 6.7% (±3% for a 95% confidence level) of the analyzed sample (18 occurrences); 13 of those have replacement messages. The direct usage of *console.** is the least present: 11 occurrences (4.1%, ±2.4% for a 95% confidence level), from which 10 have clear replacement messages to aid developers.

Finally, out of 268 cases, 75 could not be categorized and need further investigation. Additionally, 27 do not indicate deprecation and, thus, they are considered false positives.

V. Discussion

We shortly discuss our results in the light of the proposed research questions.

A. ***RQ1***. *How Do Developers Deprecate JavaScript APIs?*

Overall, the analyzed sample suggests that deprecation adoption in not frequent in JavaScript APIs. In this study, only 3% of all analyzed files contain occurrences of deprecation. Moreover, JavaScript projects deprecate their API using deprecation utilities, often throwing console warnings. This mechanism represented 32.8% of the studied sample. Using comments is also a common practice: considering both JSDoc and general code comments together, they represent 18.3%.

Despite recommendations on the Web for the use of JSDoc deprecation annotations as being a good practice, only 22 occurrences (8.2%) of JSDoc have been found in this study. This result suggests that, in practice, JSDoc might not be commonly used for deprecation purposes. We can note, however, that JavaScript developers in general prefer deprecation warnings over deprecation code comments mechanisms. We believe this is due to the fact that developers might be more prompt to notice console messages than code comments on dependent API. This hypothesis can be validated on a future work that evaluates the motivations behind the choice of a deprecation mechanism over another.

B. ***RQ2***. *Are JavaScript APIs Deprecated with Replacement Messages?*

From the categorized deprecation occurrences, we find that about 67% have replacement messages to aid developers when migrating APIs. However, those replacement messages are more common when the message is output to a console. Replacement messages in code comments have a lower occurrence rate.

To summarize, we can learn that there is no standard approach to deprecate JavaScript API, nor there is a single mechanism that is primarily used. Instead, we observe a few different approaches that are used alone or combined. This work can be further extended to evaluate each observed mechanism from developers perspectives in order to understand the reasoning behind the choice of an API deprecation approach. That can also lead to the proposal of a set of guidelines on JavaScript API deprecation best practices that help and improve the development experience.

VI. Threats to Validity

The used JavaScript parsing tool implementation might have errors that have been missed during its development. However, since it was implemented based on Flow, a well known and highly adopted parser in JavaScript community, the risk of this thread is reduced.

To identify deprecation occurrences, we only used one deprecation keyword: deprecated. Although this choice was deliberate in order to focus our research on the most used word for depreciation, this might cause us to miss other deprecation cases where that keyword was not used. Additionally, the categorization of the deprecation mechanism is subjected to the author/interpreter bias, although other members of our group verified the categories.

We focused on the analysis of 50 JavaScript open-source projects. These systems are hosted in GitHub and npm, the most popular code repository and JavaScript package manager. Despite these observations, our findings cannot be directly generalized to other systems, specifically to systems implemented in other programming languages or commercial ones. Additionally, we focused on analyzing open-source projects that have a large number of dependent clients, as we expect them to be good examples of well maintained and documented projects. Those projects are considered representative case studies since we look at open-source projects that have many dependent clients. However, they may not represent the whole population of JavaScript projects. Future replications should address these issues.

VII. Related Work

Sawant et al. [8], [9], [14]–[16] conducted several studies to investigate Java API deprecation practices. The authors assessed the impacts, the needs, the reasons, and the patterns of API deprecation. They observed that the Java deprecation mechanism does not address all developers needs when it comes to deprecation [14]. The authors also detected that Javadoc is not sufficient to understand the reasons behind

deprecation occurrences; by mining other data sources such as source code, issue tracker data and commit history, they identified 12 reasons that trigger developers to deprecate API [15]. They verified that most API client applications do not react to deprecation. Thus, they applied a survey to gather qualitative data from developers and try to explain this behavior [16]. Robbes at al. studied deprecation in the context of the Smalltalk ecosystem [7]. Brito et al. [4] [3] investigated the use of deprecation messages in Java and C#. The authors describe that 66.7% and 77.8% of Java and C# API, respectively, are deprecated with deprecation messages and that this rate does not evolve over time. Li Li et al. [13] performed an exploratory study on Android API deprecation and identified that the Android framework is regularly cleaned-up from deprecated API and their maintainers ensure that deprecated API are commented to provide replacement messages. However, those APIs are not consistently annotated and documented and the existing documentation is not frequently updated. Many other researchers study how API evolve, measure breaking changes, and analyze their impact on client systems [2] [17]. We notice that none of these cover the JavaScript ecosystem.

VIII. Conclusion

This paper presented an initial empirical study regarding deprecation in the JavaScript ecosystem. This work can help developers to better understand JavaScript API deprecation approaches and offer guidance on which mechanisms are more appropriate to a certain project context. After manually investigating the deprecation practices of 50 popular JavaScript projects, our results suggest that the use of deprecation mechanisms in JavaScript packages is low. However, we detect five different ways that developers use to deprecate APIs: deprecation utility, code comment, JSDoc, deprecation lists, and console messages. Among these solutions, deprecation utility and code comments are the most common practices. Finally, we find that the rate of helpful message is high. In this case, we detected that 67% of the deprecations have replacement messages to help API migration.

As future work, we plan to extend this research as follows:

- Providing further insights on the reasons, motivations and impressions behind the usage of JavaScript deprecation practices and which factors have an impact on the choice of a particular deprecation mechanism. We plan to perform a survey to gather qualitative data from JavaScript developers. That can also lead to the proposal of a set of guidelines on JavaScript API deprecation best practices that help and improve developers experience.
- The manual analyses and categorization performed on the deprecation occurrences could be improved to create a tool that is able to automatically identify deprecation contexts, categorize them, alert about missing replacement messages, and suggest more appropriate deprecation approaches. We plan to implement this tool and make it available for developers.

Acknowledgments

This research was partially supported by Brazilian funding agencies: CNPq, CAPES, and FAPEMIG.

References

[1] Bogart, C., Kästner, C., Herbsleb, J., and Thung, F., "How to break an API: cost negotiation and community values in three software ecosystems", in *International Symposium on Foundations of Software Engineering (FSE)*, pp. 109–120, 2016.

[2] Brito, A., Valente, M. T., Xavier, L., and Hora, A., "You Broke My Code: Understanding the Motivations for Breaking Changes in APIs", *Empirical Software Engineering*, pp. 1–35, 2019.

[3] Brito, G,. Hora, A., Valente, M. T., and Robbes, R., "Do Developers Deprecate APIs with Replacement Messages? A Large-scale Analysis on Java Systems", in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 360–369, 2016.

[4] Brito, G., Hora, A., Valente, M. T., and Robbes, R., "On the Use of Replacement Messages in API Deprecation: An Empirical Study", *Journal of Systems and Software*, vol. 137, 306–321, 2018.

[5] Granli, W., Burchell, J., Hammouda, I., and Knauss, E., "The Driving Forces of API Evolution", in *International Workshop on Principles of Software Evolution (IWPSE)*, 2017.

[6] Moser, S. and Nierstrasz, O., "The effect of object-oriented frameworks on developer productivity", *Computer*, vol. 29, no. 9, 1996.

[7] Robbes, R., Lungu, M., and Röthlisberger, D., "How do developers react to API deprecation?: the case of a Smalltalk ecosystem", in *International Symposium on Foundations of Software Engineering (FSE)*, 2012.

[8] Sawant, A. A., Robbes, R., and Bacchelli, A., "On the Reaction to Deprecation of 25,357 Clients of 4+1 Popular Java APIs", in *International Conference on Software Maintenance and Evolution (ICSME)*, pp. 400–410, 2016.

[9] Sawant, A. A., Robbes, R., and Bacchelli, A., "On the reaction to deprecation of clients of 4 + 1 popular Java APIs and the JDK", *Empirical Software Engineering*, vol. 23, 2158–2197, 2018.

[10] Tourwé, T. and Mens, T., "Automated support for framework-based software", in *International Conference on Software Maintenance*, pp. 148–157, 2003.

[11] Uddin, G., Dagenais, B., and Robillard, M. P., "Analyzing temporal API usage patterns", in *International Conference on Automated Software Engineering (ASE)*, 456–459, 2011.

[12] npm passes the 1 millionth package milestone! What can we learn?, https://snyk.io/blog/npm-passes-the-1-millionth-package-milestone-what-can-we-learn. Last access: Nov, 2019.

[13] Li, L., Gao, J., Bissyandé, T. F., Ma, L., Xia, X. and Klein, J., "Characterising deprecated Android APIs", in *International Conference on Mining Software Repositories (MSR)*, 254–264, 2018.

[14] Sawant, A. A., Aniche, M., van Deursen, A. and Bacchelli, A., "Understanding Developers' Needs on Deprecation as a Language Feature", in *International Conference on Software Engineering (ICSE)*, pp. 561–571, 2018.

[15] Sawant, A. A., Huang, G., Vilen, G., Stojkovski, S., and Bacchelli, A., "Why are Features Deprecated? An Investigation into the Motivation Behind Deprecation", in *International Conference on Software Maintenance and Evolution (ICSME)*, pp. 13–24, 2018.

[16] Sawant, A. A., Robbes, R., and Bacchelli, A., "To react, or not to react: Patterns of reaction to API deprecation", *Empirical Software Engineering*, vol. 24, pp. 3824–3870, 2019.

[17] Xavier, L., Brito, A., Hora, A., and Valente, M. T., "Historical and Impact Analysis of API Breaking Changes: A Large Scale Study", in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 138–147, 2017.